How to Train A-to-B and B-to-A Neural Networks So That the Resulting Transformations Are (Almost) Exact Inverses

Paravee Maneejuk[1], Torben Peters[2],
Claus Brenner[2], and Vladik Kreinovich[3]
[1]Faculty of Economics, Chiang Mai University
Chiang Mai, Thailand, Mparavee@gmail.com
[2]Institute of Cartography and Geoinformatics
Leibniz University of Hannover, Hannover, Germany
peters@ikg.uni-hannover.de, Claus.Brenner@ikg.uni-hannover.de
[3]Department of Computer Science, University of Texas at El Paso
El Paso, Texas 79968, vladik@utep.edu

1. Need for A-to-B and B-to-A Transformations

- In many practical problems, there are two (or more) different representations of a state, so that:
  - some operations are easier to perform in one representation, while
  - other operations are easier to perform in a different representation.
- A well-known historical case is the use of logarithms in a slide rule.
- Normally, a positive real number x is represented by two points at distance x (or proportional to x).

2. A-to-B and B-to-A Transformations (cont-d)

- In this representation, it is easy to perform additions and subtractions; however:
  - to perform multiplication or division,
  - it is better to represent each number x in a logarithmic scale, as the interval of width $\ln(x)$.
- In this case, e.g., multiplication $a, b \to a \cdot b$ can be efficiently performed as follow:
  - first, we transform both inputs a and b into the log scale, computing $a' = \ln(a)$ and $b' = \ln(b)$;
  - then, we add the results a' and b' of this transformation, thus computing $c' = a' + b'$;
  - finally, we apply the inverse transformation to c', i.e., find c for which $\ln(c) = c'$ (i.e., $c = \exp(c')$).

3. A-to-B and B-to-A Transformations (cont-d)

- One can easily see that

$$c = \exp(c') = \exp(a' + b') = \exp(a') \cdot \exp(b') = a \cdot b.$$

- So we indeed get the desired product.
- Computing the ratio a/b is similar, the only difference is that:
  - instead of adding a' and b',
  - we compute their difference $c' = a' - b'$.
- Such situations are ubiquitous, let us just name a few cases.
- In fluid mechanics, there are two alternative representations of dynamics:
  - Euler representation and
  - Lagrange representations.

4. A-to-B and B-to-A Transformations (cont-d)

- In the Euler representation, we describe how the quantities depend on time and on spatial coordinates.
- In this representation, when a particle moves, its coordinates change.
- In the Lagrange approach, we "tag" the moving particles so that:
  - when a particle moves, its coordinates remain the same,
  - but, e.g., the distance between particles changes.
- In quantum physics, we can have the Schroedinger representation, in which the state of the systems changes.

5. A-to-B and B-to-A Transformations (cont-d)

- We can also use the Heisenberg representation, in which:
  - the state of the particle remains the same, but
  - the operators corresponding to quantities (coordinates or momentum) change.
- In optics:
  - sometimes it is more convenient to represent light as particles, and
  - in other problems, it is more convenient to represent it as a wave.
- In cartography:
  - some ways of representing the Earth surface by a map provide better description of angles,
  - others better description of areas, etc.

6. A-to-B and B-to-A Transformations (cont-d)

- In signal processing, sometimes it is more convenient to describe how the signal changes with time.
- E.g., when we want to compute the largest possible deviation from the ideal signal.
- On the other hand, for filtering (and for data processing in general):
  - it is often more efficient to apply the Fourier transform and thus,
  - use the corresponding frequency representation.

7. Need for Machine Learning

- Sometimes, the transformations are straightforward.
- For example, the transformation between different maps of the same area is described by explicit formulas.
- However, in many practical situations:
  - the exact implementation of the corresponding transformations
  - can be very time consuming.
- A good example of such transformations are transformations between Euler and Lagrange coordinates.
- These transformations require solving a complex system of partial differential equations.

8. Need for Machine Learning (cont-d)

- This is especially important for time-critical applications:
  - where we need to finish computations before a deadline,
  - e.g., for predicting tomorrow's weather.
- In such situations, a natural way to drastically decrease computation time is to take into account that:
  - in practice, the values of the quantities come from measurements and
  - are, thus only known with some reasonable accuracy – usually, around 1-10%.
- Thus, there is no need to compute the answer with 10 or 13 digits after the period.

9. Need for Machine Learning (cont-d)

- It makes sense to replace:
  - the original time-consuming practically exact computations with
  - faster approximate ones, that would provide an answer with the corresponding accuracy.
- An efficient way to come with such an approximation is to use machine learning.
- In this approach:
  - several times, we run the original exact model on different inputs, and then
  - we use the corresponding results to train a machine learning algorithm – e.g., a neural network.

10. Need for Machine Learning (cont-d)

- This training may take some time.
- However, once we freeze the weights, neural-network computations become very fast.
- This idea has been efficiently applied to many real-life problems.
- It indeed allows us to drastically reduce computation time.

11. Traditional Methodology of Using Machine Learning and Its Limitations

- When we have two different representations – let us denote them A and B – we need both:
  - A-to-B transformations, and
  - B-to-A transformations.
- It is reasonable to replace both transformations by appropriately trained neural networks.
- For this purpose:
  - we start, e.g., with a large number of different states $a_1, \ldots, a_n$ in the A-representation, and
  - we use the exact A-to-B algorithm to find the corresponding B-states $b_1, \ldots, b_n$.

12. Traditional Methodology (cont-d)

- Then:
  - we train the A-to-B neural network on patterns (a_i, b_i) with input a_i and output b_i, and
  - we train the B-to-A neural network on patterns (b_i, a_i) with input b_i and output a_i.
- A neural network provides only an approximation to the actual transformation.
- It is Ok if we apply the neural network only once.
- In this case:
  - if we can select the approximations to be more accurate that the measurement accuracy,
  - the resulting inaccuracy will be negligible in comparison with the measurement inaccuracy.

13. Traditional Methodology (cont-d)

- However, in many data processing algorithm, we need to constantly switch between different representations.
- For example, in signal and image processing, we often have an iterative algorithm that:
  - switches all the time
  - between the time and frequency domains.
- In this case:
  - if we replace each exact transformation with an approximate one,
  - every time we apply a transformation, we add an extra approximation error.
- When we apply A-to-B and B-to-A transformations many time, the resulting errors accumulate.
- So, we may end up with a very inaccurate result.

14. What We Need and What We Do in This Talk

- It is thus desirable to make sure that the A-to-B and B-to-A transformations are (almost) exactly inverses:
  - if we first apply the A-to-B neural network to some input state a, and
  - apply the B-to-A neural network to the resulting state b,
  - we should get the state a back.
- The need for this inversion is especially important in economic and financial applications.
- In such applications, A-to-B and B-to-A transformations may describe:
  - options from classes A and B
  - that customers perceive as equivalent ones.

15. What We Do in This Talk (cont-d)

- In this case:
  - if a trader gets a state a' which should be equivalent to a but is actually slightly different,
  - e.g., slightly better than a,
  - we get an undesirable *arbitrage* phenomenon,
  - when a trader can earn huge amounts of money by exploiting this seemingly minor difference.
- In this talk, we describe:
  - how we can train A-to-B and B-to-A neural networks
  - so that the resulting transformations are (almost) exact inverses.

16. What We Do in This Talk (cont-d)

- In literature, there is an alternative solution:
  - using *invertible* neural networks,
  - i.e., networks in which each layer can be inverted.
- If this is how we train the A-to-B network, then:
  - by simply inverting each layer,
  - we will indeed get a B-to-A neural network
  - for which the resulting transformations are (almost) exact inverses.
- However, the restriction to invertible neural networks may make the training of a neural network less efficient.
- Indeed, the current successes of neural networks are based on non-invertible neural networks.
- With this in mind, we believe that it is better not to restrict the type of neural networks.

17. Our Proposal

- On the first stage of our proposal, we train the A-to-B network the same way as usual. Namely:
  - we start with a large number of different states $a_1, \ldots, a_n$ in the A-representation,
  - we use the exact A-to-B algorithm to find the corresponding B-states $b_1, \ldots, b_n$, and then
  - we train the A-to-B neural network on patterns (a_i, b_i) with input a_i and output b_i.
- On the second stage, we train the B-to-A network.
- The main difference from the usual approach is that, to train this network:
  - in addition to the sample (b_i, a_i) obtained on the first two sub-stages of the first stage,
  - we also use other patterns.

18. Our Proposal (cont-d)

- To be precise, here is what we suggest.
- Once the A-to-B network is trained, we generate more examples of A-states $a_{n+1}, \ldots, a_N$ $(N \gg n)$.
- To each of these new examples a_j, we apply the A-to-B network and record the corresponding B-state b_j.
- The A-to-B network is much faster than the exact A-to-B transformation that we approximating; so:
  - during the same time as the second sub-stage of the first stage,
  - we can process much more examples $(N \gg n)$.
- Finally, to train the B-to-A network, we use *both*:
  - the patterns (b_i, a_i) generated on the first stage and
  - the newly generated patterns (b_j, a_j).

19. Why It Works

- In the original method, the A-to-B and B-to-A networks are exact inverses only on n patterns (a_i, b_i).
- On all other inputs $a \neq a_i$:
  - if we first apply the A-to-B network and then the B-to-A network,
  - we, in general, do not get the same original state back.
- To be more precise, the closer a to one of a_i, the closer the result of the back-and-forth transformation to a.
- The larger n, the denser are the states a_i in the class of all possible A-states; thus, in general:
  - the smaller the distance from a to the nearest point a_i,
  - the closer the back-and-forth result to the original state a.

20. Why It Works (cont-d)

- In our proposed approach, the A-to-B and B-to-A networks are exact inverses on $N \gg n$ patterns (a_j, b_j).
- Since $N \gg n$, the new states a_j are placed much denser in the class of all possible A-states; thus:
  - the distance from an A-state a to the nearest new state a_j is much smaller than
  - the distance from a to the nearest original state a_i.
- So, for the newly trained networks, for a generic A-state a:
  - the result of applying the back-and-forth to a is mich closer to the original state a
  - than for the original neural network.
- This is exactly what we wanted.

21. Comment

- In our description, we started with the A-to-B transformation; alternatively:
  - we could start with the B-to-A transformation and
  - then apply the new idea to the A-to-B transformation.
- We should start with the one which is faster:
  - if, in general, the exact A-to-B transformation is faster, we start with the A-to-B transformation;
  - if the exact B-to-A transformation is faster, we should start with the B-to-A transformation.

22. What If We Have More Than Two Different Representations?

- In some practical situations, we have more than two different representations $A^{(1)}, \ldots, A^{(K)}$, $K > 2$.
- In such situations, we need to be able to perform a transformation between each pair.
- So, we need transformations $A^{(k)} \to A^{(k')}$ for each pair $k \neq k'$.
- This is how this problem is solved now.
- We start with a large number of different states $a_1^{(1)}, \ldots, a_n^{(1)}$, e.g., in the $A^{(1)}$-representation.
- For each of these states $a_i^{(1)}$ and for each representation $k > 1$:
  - we use the exact $A^{(1)}$-to-$A^{(k)}$ algorithm
  - to find the corresponding $A^{(k)}$-states $a_1^{(k)}, \ldots, a_n^{(k)}$.

23. More Than Two Representations (cont-d)

- Then, for each pair $k \neq k'$, we train the $A^{(k)}$-to-$A^{(k')}$ neural network on patterns

$$\left(a_i^{(k)}, a_i^{(k')}\right), \quad i = 1, \ldots, n.$$

- This process has the same limitation as in the case of two representations ($K = 2$):
  - if we apply several neural networks and get back to the same representation that we started with,
  - the resulting state may be different.
- How can we make this result closer to the original state?

24. Proposed New Algorithm: First Stage

- Similar to the case $K = 2$, the first stage of the algorithm is similar to what we do in the existing scheme:
- We start with a large number of different states $a_1^{(1)}, \ldots, a_n^{(1)}$, e.g., in the $A^{(1)}$-representation.
- For each of these states $a_i^{(1)}$ and for each representation $k > 1$:
  - we use the exact $A^{(1)}$-to-$A^{(k)}$ algorithm
  - to find the corresponding $A^{(k)}$-states $a_1^{(k)}, \ldots, a_n^{(k)}$.
  - Then, for each $k > 1$, we train the $A^{(1)}$-to-$A^{(k)}$ neural network on patterns $\left(a_i^{(1)}, a_i^{(k)}\right)$, $i = 1, \ldots, n$.

25. Proposed New Algorithm: Second Stage

- On the second stage, we do the following:
- We generate more examples of $A^{(1)}$-states

$$a_{n+1}^{(1)}, \ldots, a_N^{(1)} \quad (N \gg n).$$

- To each of these new examples $a_j^{(1)}$, for each $k > 1$:
  - we apply the $A^{(1)}$-to-$A^{(k)}$ network and
  - record the corresponding $A^{(k)}$-states $a_j^{(k)}$.
- Finally, for all $k > 1$ and $k' \neq k$, to train the $A^{(k)}$-to-$A^{(k')}$ network, we use *both*:
  - the patterns $\left(a_i^{(k)}, a_i^{(k')}\right)$ generated on the first stage and
  - the newly generated patterns $\left(a_j^{(k)}, a_j^{(k')}\right)$.

26. Acknowledgments

This research was supported by:

- the Center of Excellence in Econometrics, Chiang Mai University, Thailand;
- the German Research Foundation (DFG) as a part of the Research Training Group i.c.sens (grant GRK2159),
- the Institutes of Cartography and Geoinformatics and of Geodesy of the Leibniz University of Hannover, and
- the US National Science Foundation grants 1623190 and HRD-1242122.

This paper was written when V. Kreinovich was visiting the Leibniz University of Hannover.